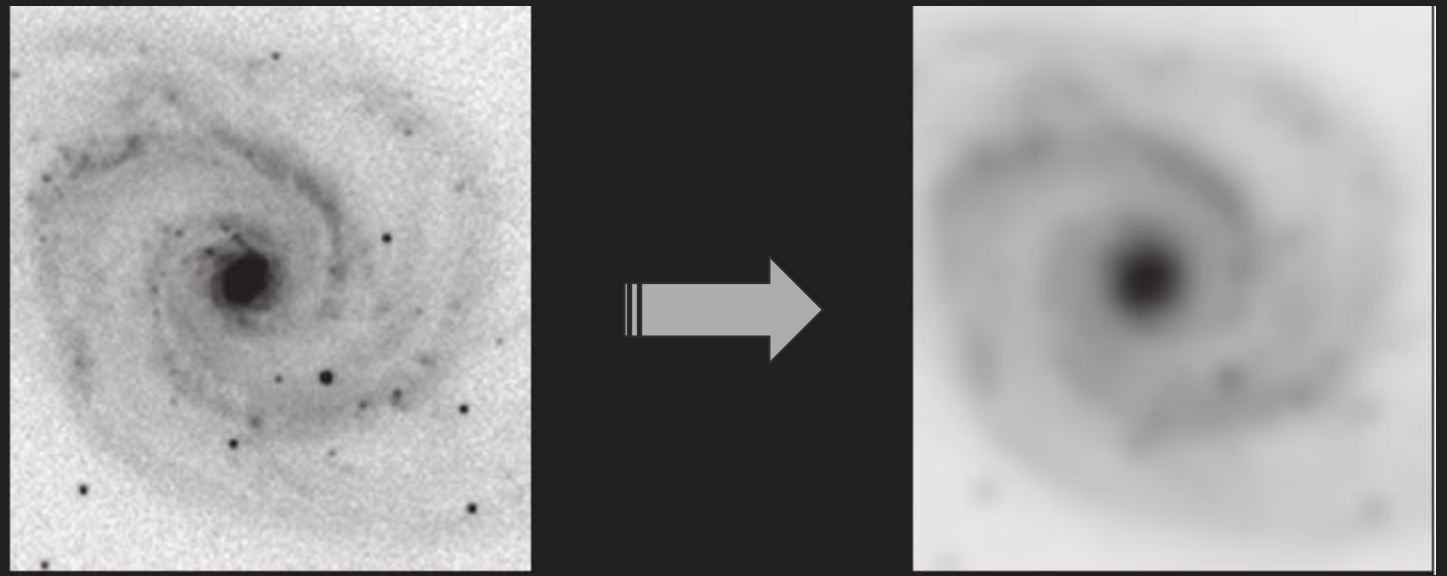
# Astronomical Image Convolution

Christian Harris

# Convolution

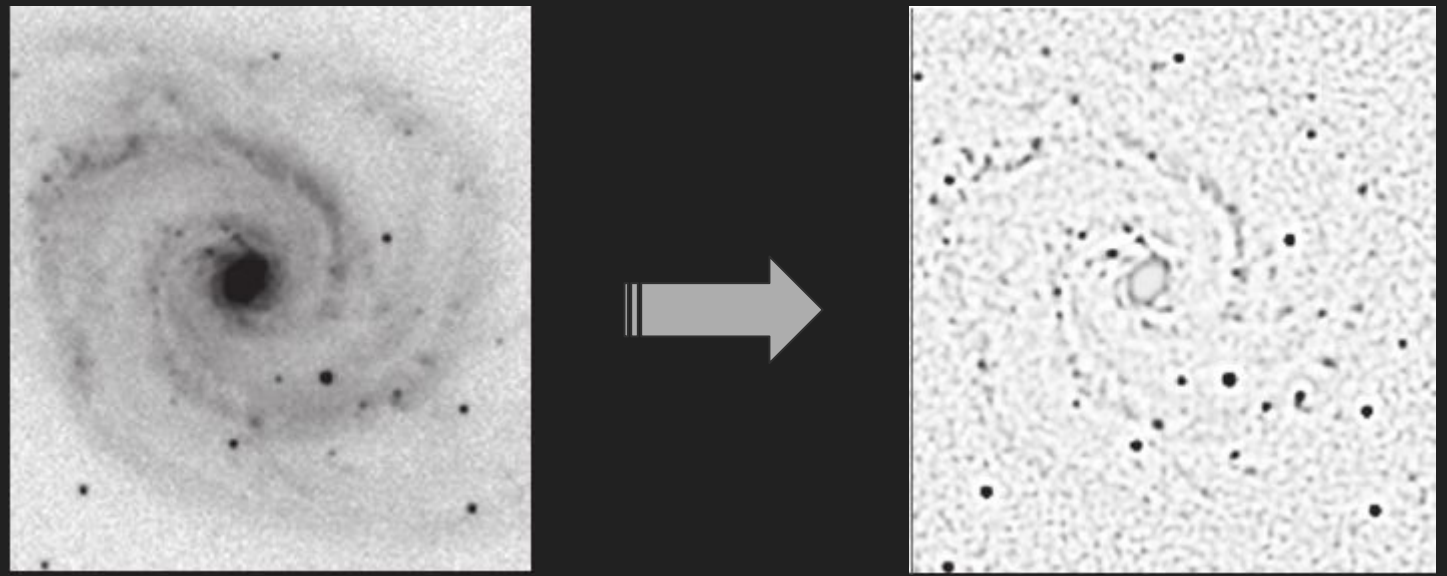
- Intentional blurring of an image to highlight specific features



Large Scale Features

# Convolution

- Intentional blurring of an image to highlight specific features

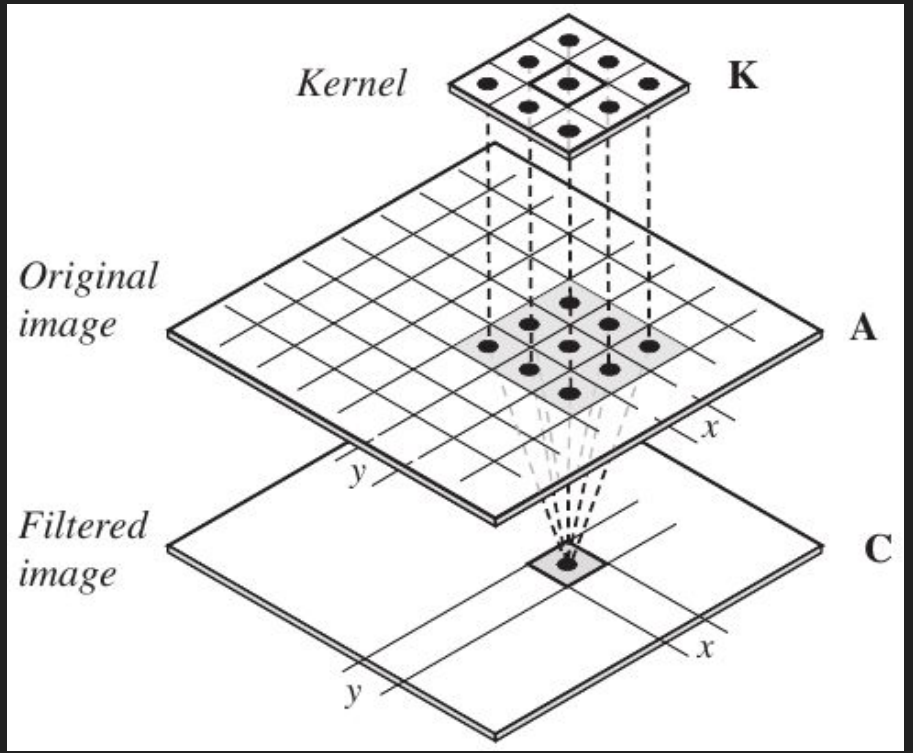


Small Scale Features

# Convolution

Combining neighboring pixels to produce a single value representative of that region

- ***Kernel*** determines how neighboring values are related



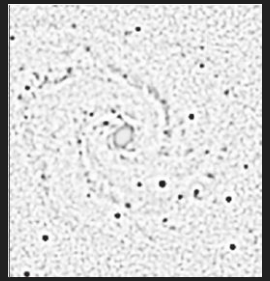
# Kernel Types

**Boxcar**

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

**Laplacian**

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |



**Gaussian**

| | | |
|---|---|---|
| 0.21 | 0.29 | 0.21 |
| 0.29 | 0.38 | 0.29 |
| 0.21 | 0.29 | 0.21 |

# Computer Specs

- Kubuntu 20.04
- Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- 4 physical cores
- 8 threads

# OpenCilk

## Compiling and running Cilk

Compiling a Cilk program is similar to compiling an ordinary C or C++ program. To compile a Cilk program using Tapir/LLVM, add the `-fcilkplus` flag to the `clang` or `clang++`

### Porting Cilk Plus code to OpenCilk

To port a Cilk Plus program to OpenCilk, once all uses of unsupported features have been updated, make the following changes to your build process:

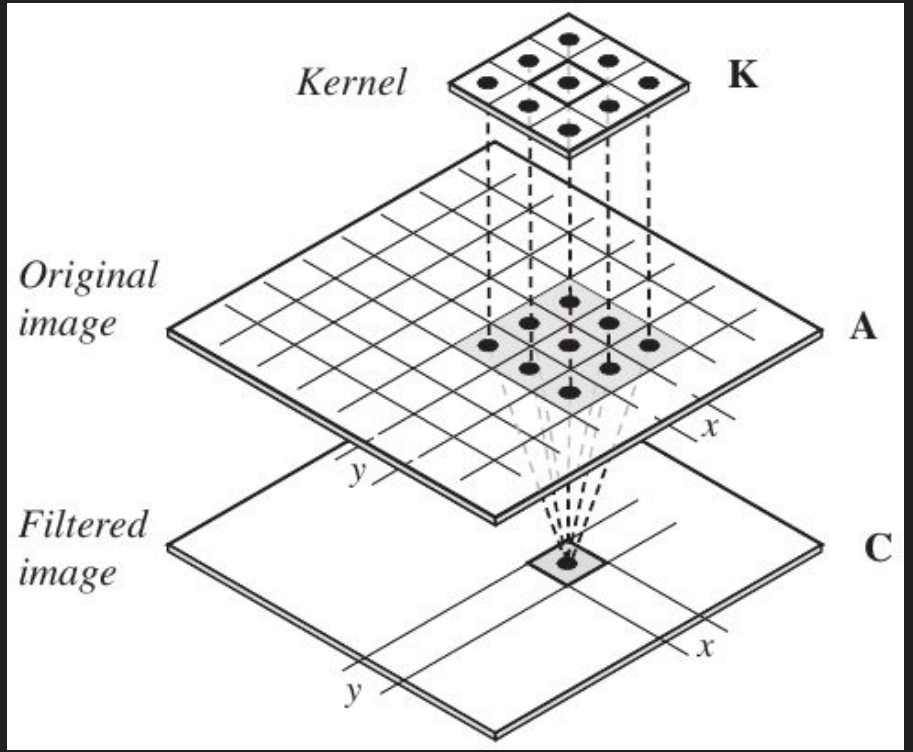
- When compiling the program, replace any uses of `-fcilkplus` with `-fopencilk`.
- When linking the program, replace any uses of `-lcilkrts` with `-fopencilk`.

- Installed from source
  - ~ 45 minutes
- First attempt crashed at 12%
  - Documented mismatch in gcc and g++ compiler version
- Compiler issues
  - Regular clang++ didn't contain OpenCilk functionality
  - Had to call clang++ from the OpenCilk build directory
- Documentation mismatch
  - -fopencilk

# Overview

- Scanning over an image
- At each pixel, apply the kernel
  - Multiply each kernel element with the associated image pixel
  - Sum
  - Assign value to new image pixel

$$C[x,y] = \sum_{i=1}^{(2V+1)} \sum_{j=1}^{(2W+1)} K[i,j]A[(x-V-1+i),(y-W-1+j)]$$



# Single Pixel Convolution

```
int convolvePixel(int x,int y, Matrix& image, Matrix& kernel) {
    int kernelLength = kernel.getRow();
    int V = (kernelLength - 1)/2;

    int counter = 0;
    int convolutionVal = 0;
    // Center the Kernel above the Image pixel location matching the NewImage pixel location
    for (int i = 0; i < kernelLength; i++) {

        for (int j = 0; j < kernelLength; j++) {
            // Current Image pixel being applied to ConvolutionVal
            int imagX = x-V+i;
            int imagY = y-V+j;

            // Exclude indices outside Image (edge detection)
            if ( ((imagX >= 0) && (imagX < image.getRow())) &&
                 ((imagY >= 0) && (imagY < image.getCol())) )
            {
                counter += kernel.get(i,j); // Track Kernel values used (flux conservation)
                convolutionVal += kernel.get(i,j) * image.get(imagX, imagY);
            }
        }
    }
    return convolutionVal / counter;
}
```

# Sequential Convolution

```
void sequentialConvolve(int startX, int endX, int startY, int endY, Matrix& convolvedImage, Matrix& originalImage, Matrix& kernel) {
    for (int i = startX; i < endX; i++) {
        for (int j = startY; j < endY; j++) {
            convolvedImage.set(i,j, convolvePixel(i,j,originalImage,kernel));
        }
    }
}
```

- Takes a region to convolve as input
- Calls convolvePixel() for each pixel in the region

# Parallel Convolution

```
void parallelConvolve(int startX, int startY, int xSize, int ySize, Matrix& convolvedImage, Matrix& originalImage, Matrix& kernel, int stopX, int stopY) {
    if ((xSize <= stopX) && (ySize <= stopY)) {
        sequentialConvolve(startX, startX+xSize, startY, startY+ySize, convolvedImage, originalImage, kernel);
    }
    else {
        cilk_spawn
            parallelConvolve(startX, startY, xSize/2, ySize/2, convolvedImage, originalImage, kernel, stopX, stopY);
        cilk_spawn
            parallelConvolve(startX, startY+ySize/2, xSize/2, ySize/2, convolvedImage, originalImage, kernel, stopX, stopY);
        cilk_spawn
            parallelConvolve(startX+xSize/2, startY, xSize/2, ySize/2, convolvedImage, originalImage, kernel, stopX, stopY);
        cilk_spawn
            parallelConvolve(startX+xSize/2, startY+ySize/2, xSize/2, ySize/2, convolvedImage, originalImage, kernel, stopX, stopY);
        cilk_sync;
    }
}
```

- Divide and Conquer subproblems of size n/2
- Assumes images are not square
- Stopping size is a parameter

# Testing Objectives

- 3x3 Boxcar kernel (all ones)
- Typical detector sizes
  - 1024x1024
  - 2048x2048
  - 4096x4096
  - 8196x8196
- Typical pixel values
  - 16-bit unsigned int -> [0, 65535]
  - Measured in analog digital units

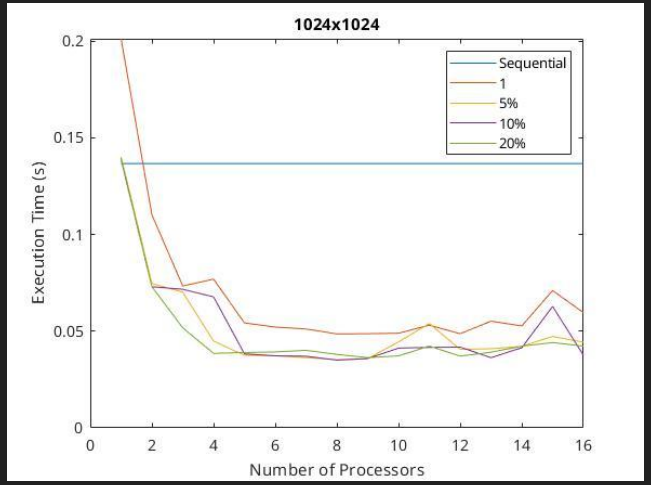
- Varying stopping sizes
  - 1
  - 5% original size
  - 10% original size
  - 20% original size

# Testing

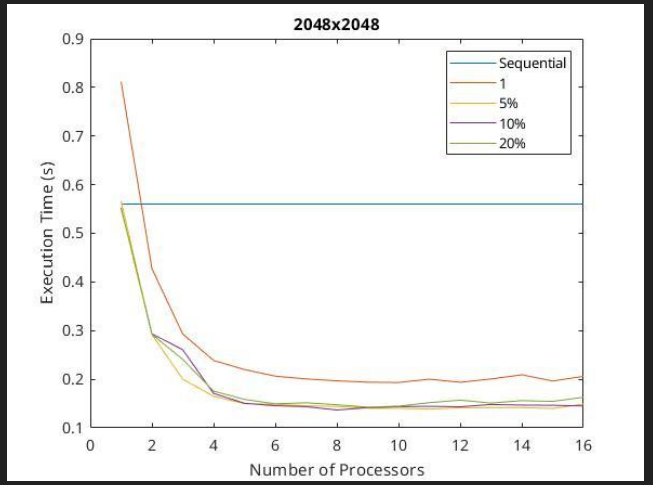
```
int n = 8192;
dataSource.open("Image_8192.txt");
Matrix image = Matrix(dataSource, n);
dataSource.close();

Matrix newPimage = Matrix(n);
int stopSize = 1;

auto start = std::chrono::high_resolution_clock::now();
parallelConvolve(0, 0, newPimage.getRow(), newPimage.getCol(), newPimage, image, kernel, stopSize, stopSize);
auto stop = std::chrono::high_resolution_clock::now();

auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
std::cout << "Parallel:    " << duration.count() << " microseconds\n";
```

- Run multiple times changing CILK_NWORKERS

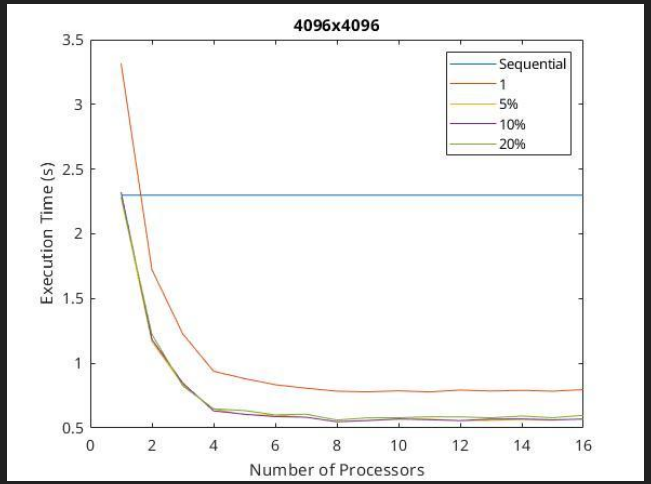
1024



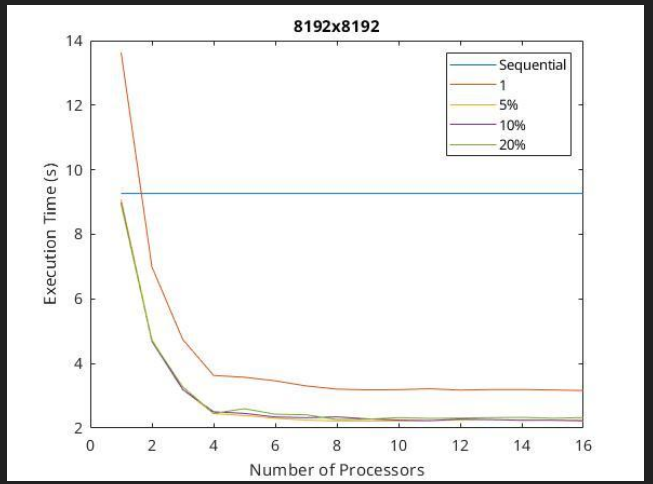

2048

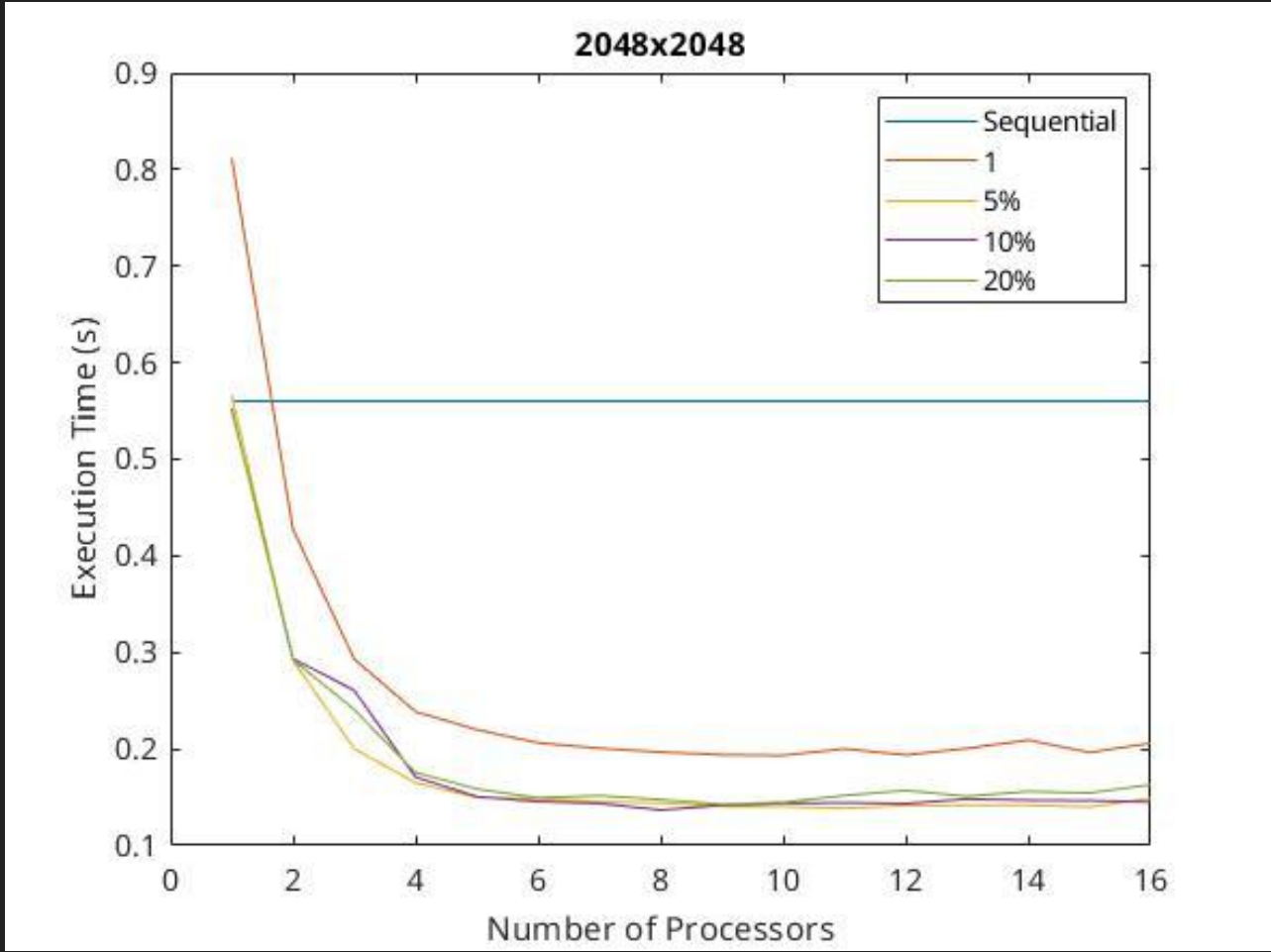



4096

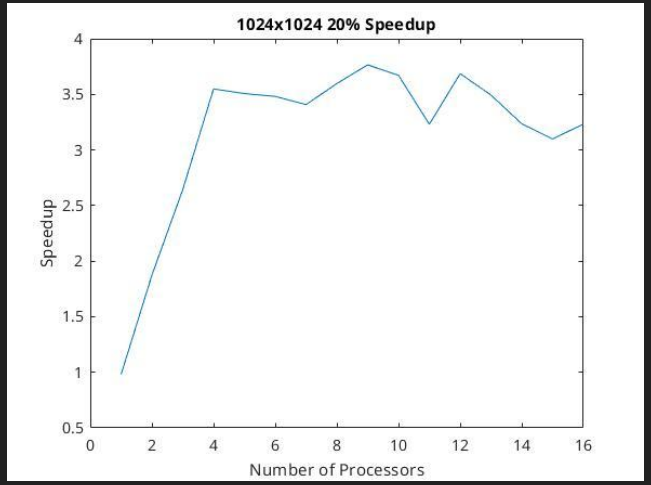



8192

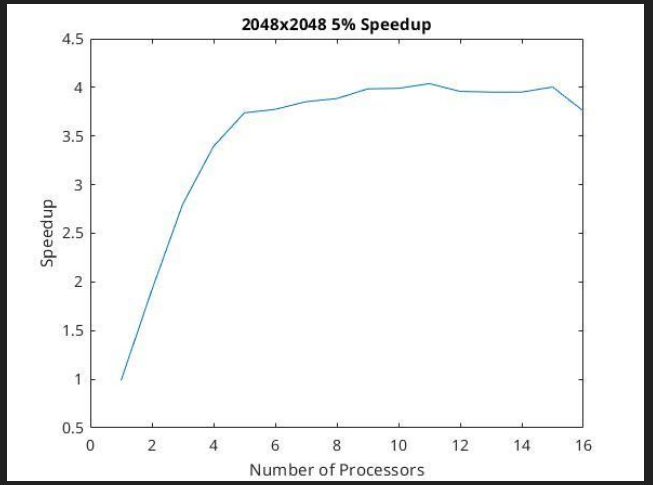



2048x2048
Execution Time (s)
Number of Processors
Sequential
1
5%
10%
20%
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
14
16

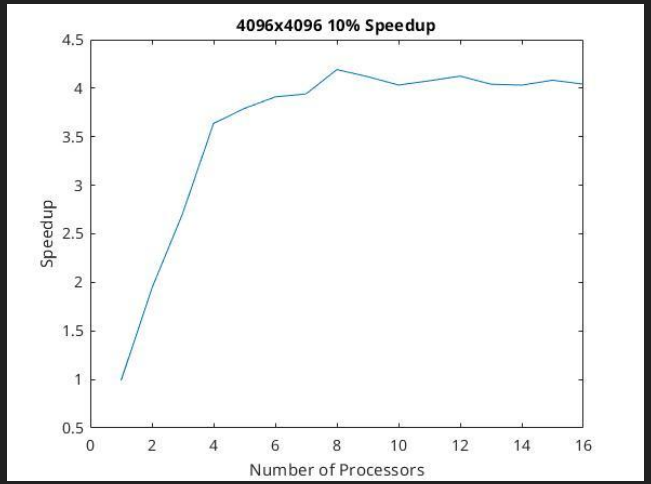
1024



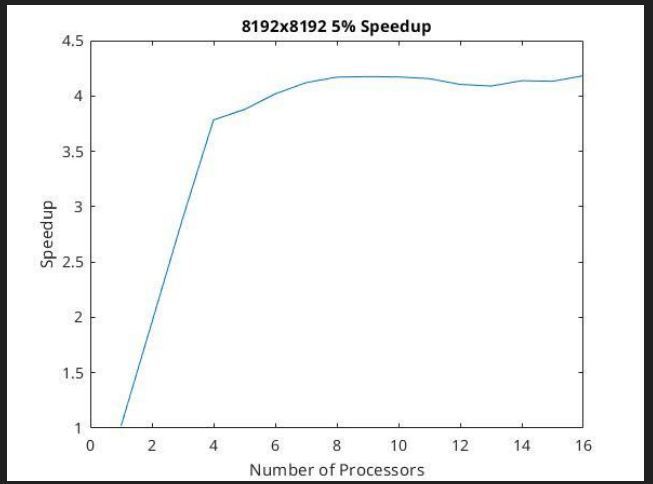

2048

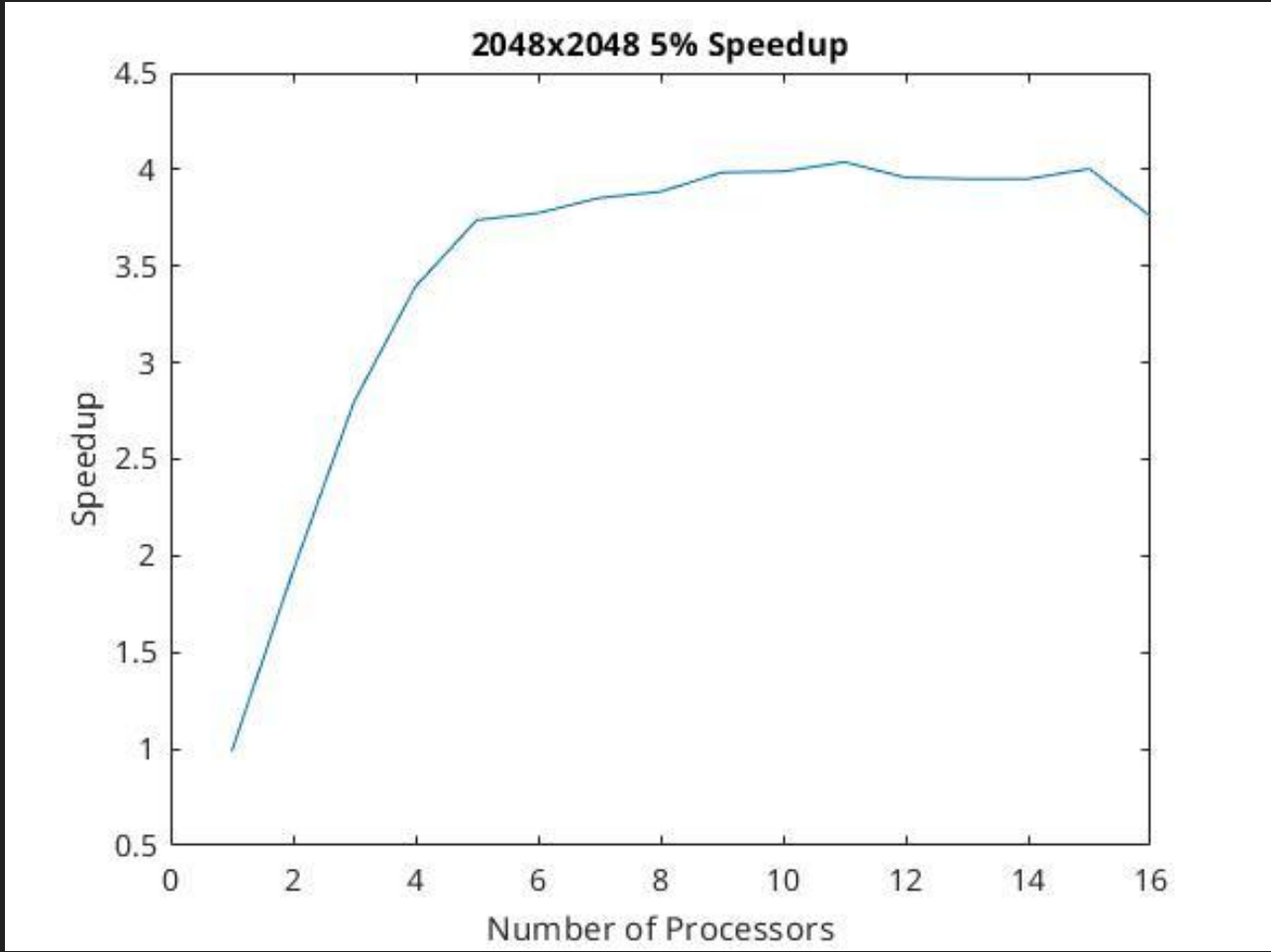



4096




8192




2048x2048 5% Speedup
Speedup
Number of Processors
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
2
4
6
8
10
12
14
16

# Analysis

- Significant decrease in execution time by not subdividing problem to size 1
- BUT, why no variation as stopping size increases?
  - Larger subproblems done sequentially, yet same execution time
  - Bug in code?
- Plateau at 4 processors
  - Gains essentially stop after physical cores exhausted

# Thank You

Questions?

Images used from Gregory Rudnick's ASTR596 Lecture 25